

LAPORAN PRAKTIKUM STRUKTUR DATA
ALGORITHM SORTING PADA JAVA BAGIAN 2



DOSEN PENGAMPU :

Dr. Wahyudi, M.T.

OLEH :

RIFKI YULIANDRA

NIM.2311532011

DEPARTEMEN INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS ANDALAS
PADANG

2024

Algorithm Sorting Pada Java Bagian-2

I. TUJUAN

- 1.1. Memahami apa yang dimaksud dengan Algoritma Sorting.
- 1.2. Memahami cara membuat program sorting menggunakan metode ShellSort
- 1.3. Memahami cara membuat program sorting menggunakan metode QuickSort.
- 1.4. Memahami cara membuat program sorting menggunakan metode MergeSort.

II. TEORI

Dalam dunia pemrograman, mengelola data secara efektif merupakan hal yang krusial. Salah satu teknik penting untuk mencapai hal ini adalah dengan menggunakan algoritma pengurutan (sorting). Algoritma pengurutan memungkinkan kita untuk menyusun data berdasarkan kriteria tertentu, seperti urutan numerik, alfabet, atau tanggal. Hal ini sangat bermanfaat untuk berbagai aplikasi, seperti analisis data, pencarian informasi, dan visualisasi data.

Java, sebagai salah satu bahasa pemrograman yang populer, menyediakan berbagai macam algoritma pengurutan yang dapat digunakan untuk menyelesaikan berbagai permasalahan. Masing-masing algoritma memiliki karakteristik dan performanya sendiri, sehingga pemilihan algoritma yang tepat sangat penting untuk mengoptimalkan performa program.

Mempelajari teori di balik algoritma pengurutan di Java bukan hanya penting bagi programmer profesional, tetapi juga bagi para pemula yang ingin memahami cara kerja program secara mendalam. Dengan memahami teori ini, programmer dapat:

- a. Memilih algoritma yang tepat: Mengetahui kelebihan dan kekurangan masing-masing algoritma memungkinkan programmer untuk memilih algoritma yang paling efisien untuk kebutuhan program mereka.
- b. Menganalisis performa program: Memahami bagaimana algoritma pengurutan bekerja membantu programmer untuk menganalisis performa program dan mengidentifikasi potensi bottleneck.
- c. Mengembangkan algoritma baru: Pengetahuan tentang teori sorting dapat menjadi dasar bagi programmer untuk mengembangkan algoritma pengurutan baru yang lebih optimal untuk kebutuhan spesifik mereka.

Algoritma sorting merupakan salah satu konsep fundamental dalam ilmu komputer dan pemrograman Java. Memahami teori di balik algoritma pengurutan sangat penting

untuk memilih algoritma yang tepat, menganalisis performa program, dan bahkan mengembangkan algoritma baru. Dengan mempelajari teori ini, programmer dapat meningkatkan kemampuan mereka dalam menyelesaikan permasalahan secara efektif dan efisien.

III. LANGKAH KERJA PRAKTIKUM

3.1. Membuat program ShellSort

```
package pekan_6;

public class ShellSort {
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println("");
    }

    /* function to sort arr using shellSort */
    int sort(int arr[]) {
        int n = arr.length;
        // Start with a big gap, then reduce the gap
        for (int gap = n / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < n; i += 1) {
                int temp = arr[i];
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                    arr[j] = arr[j - gap];
                arr[j] = temp;
            }
        }
        return 0;
    }

    public static void main(String[] args) {
        int arr[] = { 12, 34, 54, 2, 3 };
        System.out.print("Data Sebelum diurutkan: ");
        printArray(arr);
        ShellSort ob = new ShellSort();
        ob.sort(arr);
        System.out.print("Data Sesudah diurutkan menggunakan ShellSort: ");
        printArray(arr);
    }
}
```

Shell Sort adalah algoritme sorting yang mengurutkan data dengan cara membagi data menjadi beberapa bagian (gap), dan setiap bagian diurutkan menggunakan algoritme sorting Insertion Sort. Setelah setiap bagian diurutkan, gap akan dikurangi sehingga semakin kecil, dan setelah gap menjadi 1, maka semua data telah diurutkan.

Fungsi `printArray()` digunakan untuk mencetak isi dari array. Fungsi `sort()` adalah fungsi utama yang digunakan untuk mengurutkan data menggunakan Shell Sort. Dalam fungsi `sort()`, variabel `n` digunakan untuk menyimpan panjang dari array.

Pada langkah awal, gap diinisialisasi dengan nilai $n/2$, dan setiap kali perulangan dilakukan, gap akan dibagi 2 hingga menjadi 1. Perulangan awal dilakukan dengan memulai dari gap sampai dengan akhir dari array. Variabel `temp` digunakan untuk menyimpan nilai saat ini dari elemen yang sedang diproses. Perulangan kedua dilakukan dengan memulai dari `i` sampai dengan akhir dari array dengan increment gap. Perulangan ketiga dilakukan dengan membandingkan elemen dengan elemen sebelumnya dengan increment gap. Jika elemen sebelumnya lebih besar dari elemen yang sedang diproses, maka elemen yang sedang diproses akan digeser ke posisi elemen sebelumnya. Setelah perulangan ketiga selesai, maka elemen yang telah disimpan dalam variabel `temp` akan disisipkan ke posisi yang sesuai. Setelah semua perulangan selesai, maka array telah diurutkan menggunakan Shell Sort.

Adapun output yang dihasilkan:

```
<terminated> ShellSort [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe (May 27, 2024, 9:15:03)
Data Sebelum diurutkan: 12 34 54 2 3
Data Sesudah diurutkan menggunakan ShellSort: 2 3 12 34 54
```

3.2. Membuat program QuickSort

```
package pekan_6;

public class QuickSort {
    static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);

        for (int j = low; j <= high - 1; j++) {
            // if current element is smaller than the pivot
            if (arr[j] < pivot) {
                // Increment index of smaller element
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return (i + 1);
    }

    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static void printArr(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

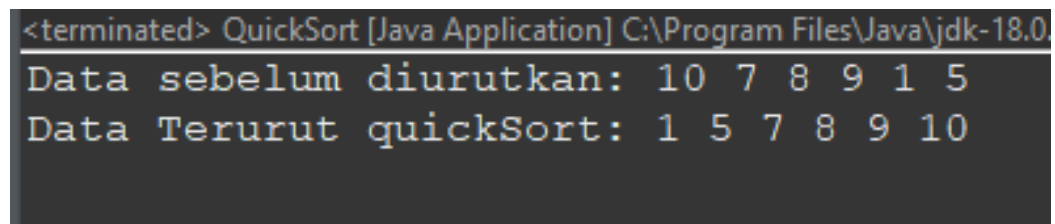
    public static void main(String[] args) {
        int[] arr = { 10, 7, 8, 9, 1, 5 };
        int N = arr.length;
        System.out.print("Data sebelum diurutkan: ");
        printArr(arr);
        quickSort(arr, 0, N - 1);
        System.out.print("Data Terurut quickSort: ");
        printArr(arr);
    }
}
```

Quick Sort adalah algoritme sorting yang bekerja dengan cara memilih sebuah pivot element, kemudian membagi data menjadi dua bagian, yaitu data yang lebih kecil dari pivot dan data yang lebih besar dari pivot. Setelah itu, algoritme akan mengulangi langkah tersebut pada setiap bagian yang telah dihasilkan hingga semua data telah diurutkan.

Fungsi `swap()` digunakan untuk memswap dua elemen pada array. Fungsi `partition()` digunakan untuk membagi data menjadi dua bagian, yaitu data yang lebih kecil dari pivot dan data yang lebih besar dari pivot. Pada fungsi `partition()`, pivot diambil dari elemen terakhir dari array. Perulangan pertama dilakukan dengan membandingkan setiap elemen dengan pivot. Jika elemen tersebut lebih kecil dari pivot, maka elemen tersebut akan dipindahkan ke depan. Setelah perulangan pertama selesai, maka elemen pivot akan dipindahkan ke posisi yang sesuai.

Fungsi `quickSort()` digunakan untuk mengurutkan data menggunakan Quick Sort. Pada fungsi `quickSort()`, perulangan dilakukan hingga `low` lebih kecil dari `high`. Pada setiap perulangan, pivot akan dipilih menggunakan fungsi `partition()`. Setelah pivot dipilih, maka Quick Sort akan dipanggil kembali pada kedua bagian yang telah dihasilkan. Fungsi `printArr()` digunakan untuk mencetak isi dari array.

Adapun output yang dihasilkan:



```
<terminated> QuickSort [Java Application] C:\Program Files\Java\jdk-18.0
Data sebelum diurutkan: 10 7 8 9 1 5
Data Terurut quickSort: 1 5 7 8 9 10
```

Hasil dari sorting akan dicetak menggunakan fungsi `printArr()`. Dengan menggunakan Quick Sort, waktu eksekusi yang dibutuhkan untuk mengurutkan data akan lebih cepat daripada sorting algorithm lainnya, seperti Bubble Sort atau Selection Sort. Hal ini disebabkan Quick Sort memiliki kompleksitas waktu yang lebih rendah, yaitu $O(n \log n)$, dibandingkan dengan Bubble Sort atau Selection Sort yang memiliki kompleksitas waktu $O(n^2)$.

3.3. Membuat program MergeSort

```
package pekan_6;

public class MergeSort {
    void merge(int arr[], int l, int m, int r) {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;
        // Create temp arrays */
        int L[] = new int[n1];
        int R[] = new int[n2];
        // Copy data to temp arrays */
        for (int i = 0; i < n1; ++i)
            L[i] = arr[l + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[m + 1 + j];
        int i = 0, j = 0;
        // Initial index of merged subarray array
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }
        // Copy remaining elements of L[] (if any) */
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }
        // Copy remaining elements of R[] (if any) */
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    void sort(int arr[], int l, int r) {
        if (l < r) {
            // Find the middle point
            int m = (l + r) / 2;
            // Sort first and second halves
            sort(arr, l, m);
            sort(arr, m + 1, r);
            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }

    // A utility function to print array of size n */
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        System.out.println("Sebelum terurut");
        printArray(arr);
        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.length - 1);
        System.out.println("Sesudah Terurut menggunakan merge Sort");
        printArray(arr);
    }
}
```

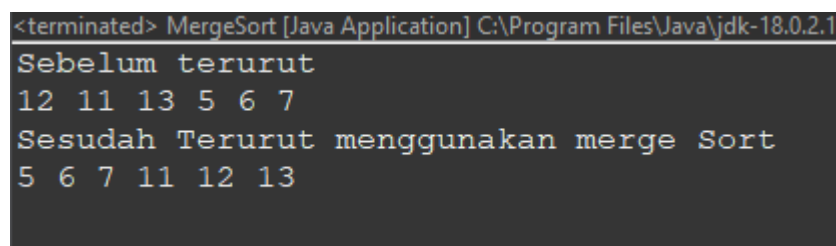
Merge Sort adalah algoritme sorting yang bekerja dengan cara membagi data menjadi dua bagian hingga setiap bagian hanya berisi satu elemen, lalu membagi kembali setiap bagian tersebut hingga semua data telah diurutkan. Setelah itu, algoritme akan menggabungkan kembali setiap bagian dengan cara membandingkan elemen-elemen pada setiap bagian.

Fungsi `merge()` digunakan untuk menggabungkan dua bagian dari array yang telah diurutkan. Pada fungsi `merge()`, dulu dibuat dua array temporer, yaitu `L[]` dan `R[]`, dengan ukuran sesuai dengan jumlah elemen pada setiap bagian yang akan diurutkan. Kemudian, elemen-elemen pada setiap bagian di copy ke array temporer tersebut. Setelah itu, dilakukan perulangan hingga semua elemen pada kedua array temporer telah diurutkan. Pada setiap perulangan, dilakukan perbandingan antara elemen pada array `L[]` dan `R[]`. Elemen yang lebih kecil akan dipilih dan dimasukkan ke posisi yang sesuai pada array asli. Perulangan akan terus dilakukan hingga semua elemen pada kedua array temporer telah dimasukkan ke array asli.

Fungsi `sort()` digunakan untuk mengurutkan data menggunakan Merge Sort. Pada fungsi `sort()`, perulangan dilakukan hingga `l` lebih kecil dari `r`. Pada setiap perulangan, pivot akan dipilih menggunakan rumus $m = (l + r) / 2$. Setelah pivot dipilih, maka Merge Sort akan dipanggil kembali pada kedua bagian yang telah dihasilkan. Setelah kedua bagian telah diurutkan, maka fungsi `merge()` akan dipanggil untuk menggabungkan kembali kedua bagian tersebut.

Fungsi `printArray()` digunakan untuk mencetak isi dari array. Pada fungsi `printArray()`, dilakukan perulangan hingga semua elemen pada array telah dicetak.

Adapun output yang dihasilkan:



```
<terminated> MergeSort [Java Application] C:\Program Files\Java\jdk-18.0.2.1
Sebelum terurut
12 11 13 5 6 7
Setelah Terurut menggunakan merge Sort
5 6 7 11 12 13
```

Hasil dari sorting akan dicetak menggunakan fungsi `printArray()`. Dengan menggunakan Merge Sort, waktu eksekusi yang dibutuhkan untuk mengurutkan data akan lebih cepat daripada sorting algorithm lainnya, seperti Bubble Sort atau Selection Sort. Hal ini disebabkan Merge Sort memiliki kompleksitas waktu yang lebih rendah, yaitu $O(n \log n)$, dibandingkan dengan Bubble Sort atau Selection Sort yang memiliki kompleksitas waktu $O(n^2)$. Namun, Merge Sort membutuhkan ruang lebih banyak daripada Quick Sort karena Merge Sort membutuhkan array temporer untuk menggabungkan kedua bagian yang telah diurutkan.

IV. KESIMPULAN

Dari ketiga program yang disajikan, kita dapat melihat bahwa setiap program mempunyai cara kerja yang berbeda-beda dalam memilah data. Program pertama yaitu Bubble Sort, bekerja dengan membandingkan pasangan elemen dalam array dan memutuskan apakah kedua elemen tersebut akan ditukar atau tidak. Proses ini akan dilakukan hingga semua elemen dalam array terurut. Program kedua yaitu Selection Sort, bekerja dengan mencari elemen terkecil pada setiap iterasi dan menyisipkannya pada posisi yang benar dalam array. Proses ini akan dilakukan hingga semua elemen dalam array terurut. Program ketiga yaitu Merge Sort, cara kerjanya adalah dengan membagi data menjadi dua bagian hingga setiap bagian hanya berisi satu elemen, kemudian membagi lagi setiap bagian tersebut hingga seluruh data terurut. Algoritma kemudian akan menggabungkan kembali setiap bagian dengan membandingkan elemen dari setiap bagian.

Dari ketiga program tersebut, kita dapat melihat bahwa merge sort lebih cepat dibandingkan bubble sort dan choice sort. Hal ini dikarenakan merge sort memiliki kompleksitas waktu yang lebih rendah yaitu $O(n \log n)$ dibandingkan bubble sort dan choice sort yang memiliki kompleksitas waktu $O(n^2)$. Namun, pengurutan gabungan membutuhkan lebih banyak ruang daripada pengurutan cepat karena pengurutan gabungan memerlukan array sementara untuk menggabungkan dua bagian yang akan diurutkan. Selain itu, pengurutan gabungan juga lebih efisien daripada pengurutan gelembung dan pengurutan pilihan untuk mengurutkan data yang lebih besar karena pengurutan gabungan memiliki waktu proses yang lebih stabil. Dengan kata lain, waktu yang dibutuhkan untuk melakukan pengurutan gabungan tidak akan bertambah secara signifikan meskipun jumlah data yang akan diurutkan menjadi lebih besar. Oleh karena itu, merge sort sering digunakan pada aplikasi yang memerlukan penyortiran data berkecepatan tinggi, seperti pencarian data di database yang besar.